Types, Arrays, Expressions in Our Programming Language

(Data types)

- **Primitive data types** are: int (integers) and bool (Boolean).
- **Composite types**: (multi-dimensional) Arrays of primitive types of values, with integer indices.
  - Our purpose is to learn about program verification, so we keep the programming language as simple as possible.

(Expressions)

- An **expression** is a piece of code who can be evaluated to some value. In our programming language, expressions have primitive type values. For example, you can consider some non-quantified predicate as an expression with Boolean value. Expressions in our programming language can be built from:
  - **Expressions**
  - **Constants**: Integers ($0, 1, -1 \dots$) and Boolean constants (T and F).
  - **Variables of primitive types**
  - **Functions** that return **primitive type values**

  - **Operations:**
    - On integers: $+,\ -,\ *,\ /,\ \%,\ =,\ \neq,\ <,\ \leq,\ >,\ \geq,\ sqrt()$...
      - ❖ Note that, / and $sqrt()$ round toward 0 to an integer. For example, $13/\ 3 = 4$, $13\ /(-3) = -4$, and $sqrt(17) =\ 4$.
      - ❖ Division and mod by 0 and sqrt of negative values generate runtime errors.
    - On Booleans: $\neg,\ \wedge,\ \vee,\ \rightarrow,\ \leftrightarrow,\ =,\ \neq$ ...
    - On arrays: $size(\ )$ and array element selection. For example, $b = (0,3,4)$, then $size(b) = 3$.

  - **Arrays:**
    - As usual, $b[e]$ is array element selection. Note that, e is an expression evaluates to an integer.
    - $size(b)$ gives the length of b.
    - In a multi-dimension array, $b[e_0][e_1] \dots [e_{n-1}]$ is selecting the element with index e_0 in the first dimension, e_1 in the second dimension … e_{n-1} in the n^{th} dimension. Note that, n is not a variable but an integer constant here. ("…" is not understandable). For example, $b = \big((6,3), (2,5,8)\big)$, then $b[1][2] = 8$.
    - Note that, we can never have an array appear by itself in an expression, it is always wrapped in some function, or we are selecting some element in the array.

  - **Conditional:** $\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}$
    - Semantically, if B evaluates to true, then evaluate e_1; if B evaluates to false, then evaluate e_2.
    - Note that, e_1 and e_2 are expression and we require them to have the same type.
    - There is also "$\textbf{if} - \textbf{else if} - \textbf{else}$" in our programing language, is written as "$\textbf{if } B_1 \textbf{ then } e_1 \textbf{ else } B_2 \textbf{ then } e_2 \textbf{ else } e_3 \textbf{ fi}$".

- Note that:
  - We don't explicitly declare variables; we assume that we can infer the types. For example: to have expression $p \vee x > 0$, we don't need something like "create variable x of type int".

- An expression must evaluate to a primitive type of value, so it cannot evaluate to an array. For example: (assuming a and b are two arrays) $\textbf{if } B \textbf{ then } a \textbf{ else } b \textbf{ fi}[0]$ is illegal. But $\textbf{if } B \textbf{ then } a[0] \textbf{ else } b[0] \textbf{ fi}$ is legal.

- Functions who return primitive type of values are allowed in an expression, but an expression cannot yield a function. For example: $\textbf{if } B \textbf{ then } f(x) \textbf{ else } g(x) \textbf{ fi}$ is legal; but $\textbf{if } B \textbf{ then } f \textbf{ else } g \textbf{ fi } (x)$ is not.

1. Are the following expressions legal?
   a. $x \,\%\, b[y]$ Yes
   b. $a[0:2]$ No
   c. $\textbf{if } x < 0 \textbf{ then } x * x \textbf{ else } sqrt(x) \textbf{ fi} + y$ Yes
   d. $\textbf{if } x < 0 \textbf{ then } p \textbf{ else } T \textbf{ fi} + y$ No
   e. $\textbf{if } i < 0 \textbf{ then } b[0] \textbf{ else } i \geq size(b) \textbf{ then } b[size(b) - 1] \textbf{ else } b[i] \textbf{ fi}$ Yes

(Notations)

- Most commonly, c and d are constants; e and s are general expressions; B and C are Boolean expressions; a and b are array names; and u, v, etc. are variables. Greek letters like α and β stand for semantic values.

(Evaluate an expression)

- In general, evaluation is a process to translate something "syntactic" to something "semantic".
- With a proper state, an expression can be evaluated to a value of primitive type.
  - For example: $\sigma = \{x = 5, y = 2\}$, then $\sigma(x * y) = 10$. Here, $x * y$ is an expression that we want to evaluate, and σ is a state that's proper for $x * y$.

- The value of $\sigma(e)$ depends on what kind of expression e is, so we use recursion on the structure of e (the base cases are variables and constants, and we recursively evaluate sub-expressions).

  - $\sigma(x) =$ the value that σ binds variable x to. For example, if $\sigma = \{x = 5\}$, then $\sigma(x) = 5$
  - $\sigma(c) =$ the value of the constant c. For example, $\sigma(5) = 5$. (σ is irrelevant here.)
  - $\sigma(e_1 + e_2) =$ the value of $\sigma(e_1)$ plus the value of $\sigma(e_2)$ [and similar for $-$, $*$, $/$ $etc.$].
  - $\sigma(e_1 < e_2) = T$ **iff** the value of $\sigma(e_1)$ is less than the value of $\sigma(e_2)$ [similar for $\leq$, $=$, etc].
  - $\sigma(e_1 \wedge e_2) = T$ **iff** the value of $\sigma(e_1)$ and the value of $\sigma(e_2)$ are both T [similar for $\vee$, $\rightarrow$ etc].
  - $\sigma(\textbf{if } B \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) = \sigma(e_1)$ if the value of $\sigma(B) = T$; it $= \sigma(e_2)$ if the value of $\sigma(B) = F$.

- As an aside, here we have a question, when we evaluate an expression, how does something "syntactic" become something "semantic"? In other words, how is a piece of code compiled into something meaningful to us? In this small section, to make this clear, I will use highlights to show the values (which are something semantic).

2. Let $z \equiv 2 + 3$, evaluate $\sigma(z)$.

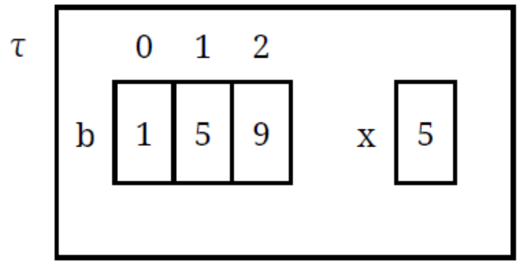
$$\sigma(2+3) = \sigma(2) + \sigma(3) = 2 + 3 = 2 + 3 = 5$$

3. Let $\sigma = \{x = 1\}$, let $\tau = \sigma \cup \{y = 1\}$, and let $e \equiv (x = \textbf{if } y > 0 \textbf{ then } 17 \textbf{ else } y \textbf{ fi})$, evaluate $\tau(e)$.

$$\begin{aligned}
\tau(x = \textbf{if } y > 0 \textbf{ then } 17 \textbf{ else } y \textbf{ fi}) &= \big(\tau(x) = \tau(\textbf{if } y > 0 \textbf{ then } 17 \textbf{ else } y \textbf{ fi})\big) \\
&= \big(1 = \tau(\textbf{if } 1 > 0 \textbf{ then } 17 \textbf{ else } y \textbf{ fi})\big) \\
&= (1 = \tau(\textbf{if } T \textbf{ then } 17 \textbf{ else } y \textbf{ fi})) \\
&= (1 = \tau(17)) \\
&= (1 = 17) \\
&= (1 = 17) = F
\end{aligned}$$

(Arrays and their values)

4. How to write the following state τ in our language?



- $\tau = \{b[0] = 1, b[1] = 5, b[2] = 9, x = 5\}$
  We take the **value of an array** to be a **function** from index values to stored values.

- $\tau = \{b = \beta, x = 5\}$ where $\beta(0) = 1,\ \beta(1) = 5,\ \beta(2) = 9$
  If we give the function a name β, then we can write τ like this.

- $\tau = \{b = \beta, x = 5\}$ where $\beta = \{(0,1), (1,5), (2,9)\}$
  The function β can also be expressed as a collection of tuples (index, stored value).

- $\tau = \{b = \beta, x = 5\}$ where $\beta = (1,5,9)$
  The function β can also be simplified to a sequence of values.

- $\tau = \{b = (1,5,9), x = 5\}$

5. Let $\sigma = \{x = 1, b = \beta\}$ where $\beta = (2,0,4)$, evaluate $\sigma(b[x+1] - 2)$.

$$\begin{aligned}
\sigma(b[x+1]-2) &= \sigma(b[x+1]) - \sigma(2) \\
&= \sigma(b)\big(\sigma(x+1)\big) - 2 \\
&= \sigma(b)\big(\sigma(x) + \sigma(1)\big) - 2 \\
&= \sigma(b)(1+1) - 2 \\
&= \beta(2) - 2 \\
&= 4 - 2 = 2
\end{aligned}$$

<u>Updating a State</u>

- For any state σ, variable x, and value α, the "**update**" of σ at x with α, written $\sigma[x \mapsto \alpha]$, is the state that is a copy of σ except that it binds variable x to value α.
  - Note that, we are not really updating σ itself (although that is the traditional way to call this operation), that's why we quote the word "update": $\sigma[x \mapsto \alpha]$ is a new state and σ is not changed.
- We can give $\sigma[x \mapsto \alpha]$ a new name but we don't have to. We read $\sigma[x \mapsto \alpha](v)$ left-to-right — we're taking the function $\sigma[x \mapsto \alpha]$ and applying it to variable v.

6. Let $\sigma = \{x = 1,\ y = 2\}$, answer the following questions about state τ.
   a. Let $\tau = \sigma[x \mapsto 3]$, then $\tau = \{x = 3,\ y = 2\}$.
   b. Let $\tau = \sigma[z \mapsto 3]$, then $\tau = \{x = 1,\ y = 2,\ z = 3\}$.
      - $\sigma(z)$ doesn't need to be defined (z is bind with a variable in σ) before updating σ.

c. Let $\tau = \sigma[x \mapsto 1]$, then $\tau = \{x = 1,\ y = 2\}$.
   - τ and σ are consist of the same bindings, they are not syntactically equivalent though (they are not the same state).

7. True or False
   a. If $\sigma(x)$ is not defined, then $\sigma[x \mapsto 0] = \sigma \cup \{x = 0\}$.
      True
   b. If $\sigma(x)$ is defined and $\sigma(x) \neq 0$, then $\sigma[x \mapsto 0] = \sigma \cup \{x = 0\}$.
      False, $\sigma \cup \{x = 0\}$ becomes ill-formed since x appears twice.
   c. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto 0] \vDash x \geq x^2$.
      Ture
   d. Let $x \not\equiv y$ be both bound in σ, then $\sigma[x \mapsto 0](y) = \sigma(y)$
      True.
   e. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto x + 1] = \{x = 6\}$
      False, we cannot bind a variable with an expression (something syntactic), it becomes ill-formed.
   f. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto 2 + 1] = \{x = 3\}$
      True, $2 + 1$ is a semantic value. Remember that a function who returns a primitive type is also semantic.
   g. Let $\sigma = \{x = 5\}$, $\sigma[x \mapsto \sigma(x + 1)] = \{x = 6\}$
      True.

- We can do a sequence of updates on a state, such as $\sigma[x \mapsto 0]\ [y \mapsto 8]$. Here, we read it left-to-right.
  - For example, let $\sigma = \{x = 2, y = 6\}$, then $\sigma[x \mapsto 0][y \mapsto 8] = \{x = 0, y = 6\}[y \mapsto 8] = \{x = 0, y = 8\}$.

8. True or False
   a. Let $x \not\equiv y$, then $\sigma[x \mapsto 0][y \mapsto 8] = \sigma[y \mapsto 8][x \mapsto 0]$
      True. The order of update doesn't matter if we have two different variables.
   b. Let $x \not\equiv y$, then $\sigma[x \mapsto 0][y \mapsto 8] \equiv \sigma[y \mapsto 8][x \mapsto 0]$
      False. Although they give the same state, the updating procedures are different.
   c. $\sigma[x \mapsto 0][x \mapsto 8] = \sigma[x \mapsto 8]$
      True. The second update supersedes the first.
   d. $\sigma[x \mapsto 0][x \mapsto 8] \equiv \sigma[x \mapsto 8]$
      False. Although they give the same state, the updating procedures are different.

9. Let $\sigma = \{x = 1\}$, then what is $\sigma[x \mapsto 2][z \mapsto \sigma[x \mapsto 3](x) + 10]$?

$$\begin{aligned}\sigma[x \mapsto 2][z \mapsto \sigma[x \mapsto 3](x) + 10] \quad &= \{x = 2\}[z \mapsto \sigma[x \mapsto 3](x) + 10] \\ &= \{x = 2\}[z \mapsto \{x = 3\}(x) + 10] \\ &= \{x = 2\}[z \mapsto 13] \\ &= \{x = 2, z = 13\}\end{aligned}$$

- How to update a value in an array? What do we do if we want to update the value in $b[0]$? Since we handle array as a function from an index to the value stored, here let's expand the notion of updating states to updating functions.
- If δ is a function and α and β are valid elements of the domain and range of δ respectively, then the update of δ at α with β, written $\delta\,[\alpha \mapsto \beta]$, is the function defined by $\delta[\alpha \mapsto \beta](\gamma) = \beta$ if $\gamma = \alpha$ and $\delta[\alpha \mapsto \beta](\gamma) = \delta\,(\gamma)$ if $\gamma \neq \alpha$.
  - Note that, if we consider state as a function, then the definition of updating a state follows the above definition as well. The only difference is that the α and γ here are values.

For example, let function $\delta = \{(4,6), (3,7), (2,5)\}$, then $\delta[2 \mapsto 3] = \{(4,6), (3,7), (2,3)\}$. Also, $\delta[2 \mapsto 3](2) = 3$, $\delta[2 \mapsto 6](3) = 7$.

- Say σ is a (proper) state with an array b, with η = the function $\sigma(b)$. If α is a valid index value for b, then $\sigma[b[\alpha] \mapsto \beta]$ means $\sigma\left[b \mapsto \eta[\alpha \mapsto \beta]\right]$. So, updating σ at $b[\alpha]$ with β involves updating σ with an updated version of η, namely $\eta\,[\alpha \mapsto \beta]$, as the value of b.
  - For example, $\sigma = \{x = 3, b = (2,4,6)\}$, then $\sigma[b[0] \mapsto 8] = \{x = 3, b = (8,4,6)\}$. Here, $\sigma(b)$ is $(2,4,6)$ as a function (which can also be written $\{(0,2), (1,4), (2,6)\}$, so $\sigma(b)[0 \mapsto 8]$ is the function $(2,4,6)[0 \mapsto 8] = (8,4,6)$.